Report

1. Youssef Amr Mohamed 46-3661 (T-20)

2. Mennatallah Ahmed 46-3693 (T-20)

3. Alia Ahmed Saddik 46-7537 (T-20)

4. Salma Khalid Shreef 46-5953 (T-20)

In this report we will explain our approach, code structure and test cases we have made and the steps we did to develop and run our code.

Step1:

We have implemented 6 classes including our main java class:

1. FPregisterFile: which is for the register file. It has 3 attributes: RegName (for the name of the class), Qi(which is the Qi for the register file), Content(value for each floating register).

2. loadBuffer: which is the load buffer where we store "load" instructions in it to be executed. It has 3 attributes: tag (which is the tag: L1,L2,L3), Busy(where if it is 0 we can store in the load buffer), Address(address we will fetch the memory with).

3. StoreBuffer: which is the store buffer where we store "store" instructions in it to be executed. It has 3 attributes: Tag(which is the tag:L1,L2,L3), Busy(where if it is 0 we can store in the store buffer), Address(address we will fetch the memory with), V(the value we will use to store in the memory), Q(if the value is not ready).

4. ReadyQueueElement: a class for instructions who got issued. It has 3 attributes: tag (tag of the instruction), latency (time for instruction to finish execution), mustWait flag (initially we set it to false whenever instruction got issued until it start to execute it will be set to true).

5. ReservationStations: reservation station itself used for both Add/Sub and Mul/Div. It has 8 attributes: tag (which is the tag: A1,A2 or M1,M2), Busy(where if it is 0 we can store in the reservation station), Op(operation), Vj (value of source register 1), Vk (value for source register 2), Qj (if the value is not ready for vj), Qk (if the value is not ready for vk), Address.

6. Tomasulo: Our main class, where we run and test our instructions with it.

Step2:

We have added a text file named Instructions where we put our Instructions we want to test with.

Step3:

We created global variables which we have used in all our methods:

1. “InstructionQueue”: a queue where we add our instructions in it
2. “ReadyQueue”: a queue where we add our instructions which got issued.
3. “WriteReady”: a queue where we add our instructions which got finished execution and ready to write on the bus.
4. “Instructionlatency”: a hashmap containts the latency the user enters for each type of instruction.
5. “RSAddSub”: array for add/sub reservation station.
6. “RSMulDiv”: array for mul/div reservation station.
7. “Stores”: array for store buffer.
8. “Loads”: array for load buffer.
9. “RegFile”: array for register file.
10. 5 counters for each of those arrays where it got incremented whenever we add in the array.
11. “cycle”: where we count our cycles.
12. “Memory”: array for the memory

Step4:

We created two methods, one for the register file to initialize it named InsertInRegFile(). And the other for the memory named memoryvalues() to initialize by storing in each index the index itself (memory[1]=1)

Step5:

We created a method named Code Parser which takes filename as a parameter and parse it line by line and each line we have one instruction where we add it in the “InstructionQueue”.

Step 6:

We created a method called issueInst(String Inst) which takes an instruction as a parameter at a time and issue it if there is a free place in it's specific reservartion station/buffer. If there is a place for it we remove it from the "InstructionQueue" and added it to it's specific global array(reservation station/buffer) and also to "ReadyQueue" which means it is issued and should be executed if it's values are ready.

Step 7:

We created a **public static void** Execute() method where it loops on the "ReadyQueue" queue and only executes each ready instruction in it once and if the instruction finished executing in this cycle it will add the instruction to "WriteReady" queue which means it will be ready to be written back in the next cycle.

Step 8:

We created a **public static void** WriteBack() method where it only writeback one of the instructions that were ready to be posted on the bus and writes back in the register file , empties the reservation station of the instruction that has been written back, and adding the instructions, that were in the reservation stations waiting for specific values to be written on the bus, to the ready queue.

Step 9:

We created a **public static void** run() method where it increments the cycles in addition to calling each of issueInst(), Execute() and writeback() every cycle so that the cycles can run smoothly.

Step 10:

For testing the whole project an Instruction file were added in the project and print statements were added in the run() method to print for each cycle the

1. Cycle Number
2. Instruction Queue
3. Ready Queue
4. Register File Contents
5. Addition and Subtraction stations
6. Multiplication and Division reservation stations
7. Load Buffer
8. Store Buffer
9. Which Instructions are executing in this cycle
10. Which Instructions finished executing in this cycle

Instructions that were used to test:

MUL.D F3 F1 F2

ADD.D F5 F3 F4

ADD.D F7 F2 F6

ADD.D F10 F8 F9

MUL.D F11 F7 F10

ADD.D F5 F5 F11